3.3 Filtering Collections

Now we're going to take a look at one of the most common steps in expressing statements in predicate logic and in processing large collections of data. At first glance these two might not appear that related, after going through this section you should be able to appreciate this elegant connection between predicate logic and data processing.

Expressing conditions in predicate logic

We saw in the last section that the universal quantifier $\forall$ is used to express a statement of the form "every element of set S satisfies ____". This works well when we use a predefined set for S (like the numeric sets $\mathbb{N}$ or $\mathbb{R}$), but does not work well when we want to narrow the scope of our statement to a smaller set.

For example, consider the following statement: "Every natural number n greater than 3 satisfies the inequality $n^2 + n \geq 20$." The phrase "greater than 3" is a *condition* that modifies the statement, limiting the original domain of n (the natural numbers) to a smaller subset (the natural numbers greater than 3).

There are two ways we can represent such conditions in predicate logic. The first is to define a new set; for example, we could define a set $S_1 = \{n \mid n \in \mathbb{N} \text{ and } n > 3\}$, and then simply write $\forall n \in S_1,~ n^2 + n \geq 20$.

The second approach is to use an implication to express the condition. To see how this works, first we can rewrite the original statement using an "if … then …" structure as follows: "For every natural number n, if n is greater than 3 then n satisfies the inequality $n^2 + n \geq 20$." We can translate this into predicate logic as $\forall n \in \mathbb{N},~ n > 3 \Rightarrow n^2 + n \geq 20$.

This works because the $n > 3 \Rightarrow$ has a filtering effect, due to the *vacuous truth* case of implication. For the values $n \in \{0, 1, 2\}$, the hypothesis of the implication, $n > 3$ is False, and so for these values the implication itself is True. And then since the overall statement is universally quantified, these vacuous truth cases don't affect the truth value of the statement.

The "forall-implies" structure is one of the most common forms of statements we'll encounter in this course. They arise naturally any time a statement is universally quantified, but there are conditions that limit the domain that the statement applies to.

Filtering collections in Python

Now let's turn our attention back to Python. Last chapter, we learned about several aggregation functions (like `sum`, `max`), and we've just learned about two more, `any` and `all`.

Sometimes, however, we want to limit the scope of one of these functions to certain values in the input collection. For example, "find the sum of only the even numbers in a collection of numbers", or "find the length of the longest string in a collection that starts with a `'D'`". For these problems, we can quickly identify which aggregation function is necessary, but the problem is in choosing the right argument to pass in.

This is where filtering appears. In programming, a **filter operation** is an operation that takes a collection of data and returns a new collection consisting of the elements in the original collection that satisfy some predicate (which can vary from one filter operation to the next).

There are different ways of accomplishing a filter operation in Python. The simplest one builds on what we've learned so far by adding a syntactic variation to comprehensions. We'll use as our example a set comprehension here, but what we'll discuss applies to list and dictionary comprehensions as well.

```
{<expression> for <variable> in <collection> if <condition>}
```

The new part, `if <condition>`, is a boolean expression involving the `<variable>`. This form of set comprehension behaves the same way as the ones we studied last chapter, except that `<expression>` only gets evaluated for the values of the variable that make the condition evaluate to `True`. Here are some examples to illustrate this:

```
>>> numbers = {1, 2, 3, 4, 5}  # Initial collection
>>> {n for n in numbers if n > 3}  # Pure filtering: only keep elements > 3
{4, 5}
>>> {n * n for n in numbers if n > 3}  # Filtering with a data transformation
{16, 25}
```

By combining these filtering comprehensions with aggregation functions, we can now achieve our goal of limiting the scope of an aggregation.

```
>>> numbers = {1, 2, 3, 4, 5}
>>> sum({n for n in numbers if n % 2 == 0})  # Sum of only the even numbers
6
```

The keyword `if` used in this syntax for filtering comprehensions is directly connected to our use of implication above. Just as we used the hypothesis $n > 3 \Rightarrow$ to limit the scope of the universal quantifier to a subset of the natural numbers, here we use `if n % 2 == 0` to limit the scope of the `sum` to just a subset of `numbers`.

Our final example in this section should make this connection even more explicit. Here's how we could translate the statement $\forall n \in S,\ n > 3 \Rightarrow n^2 + n \geq 20$ into a Python expression:

```
>>> numbers = {1, 2, 3, 4, 5, 6, 7, 8}
>>> all({n ** 2 + n >= 20 for n in numbers if n > 3})
True
```

CSC110 Course Notes Home